Serving GILS Records with Zebra

Index Data, info@index.ping.dk *Revision* : 1.5

This document explains how to set up a simple database of Government Information Locator Records using the Zebra retrieval engine and Z39.50 server (version 1.0a6 or later).

1 Introduction

Zebra is a powerful and versatile information management system which allows you to construct arbitrarily complex record structures and manage efficient, robust databases.

Since the internal data modeling tools of Zebra are based on the Z39.50-1995 standard, the system is well-suited to support a complex database profile such as the one specified by the GILS profile. Because GILS is expected by many to play an important role in the evolving, global information society, and because the GILS profile has proven useful to a number of different applications outside of its dedicated domain, the public distribution of Zebra includes a configuration set-up which makes it a simple matter to establish a new GILS database.

This document, which is a supplement to the general documentation set for Zebra, explains in simple terms how you can easily set up your own GILS-compliant database service.

2 Retrieving and Unpacking Zebra

The first step is to download the software. If you are using a WWW browser, you can point it at the Zebra distribution archive at <`http://www.indexdata.dk/zebra.html`>, and follow the link named *Download the latest version of the software (xxx)*, where *xxx* is the current version of Zebra.

If you use an FTP client, you can use normal, anonymous FTP. Connect to the host `ftp.indexdata.dk`, log in as `ftp`, and give your Email-address as the password. Then type `cd index/yaz`, and use the `dir` command to locate the current version of Zebra. The file will be named `zebra-xxx`, where `xxx` is the current version of the software. Remember to use the `bin` command before using `get` to download the software.

Once the distribution archive has been dowloaded, it must be decompressed. To do this, use the command `gunzip` command (if your system doesn't have the `gunzip` program, you will need to acquire this separately). Finally, use the command `tar xvf` <`file`> to unpack the archive.

If you downloaded the source version of the software (this is the only option today, although we expect to release binary versions for Linux, SunOS, and Digital Unix shortly), you will have to compile Zebra before you can use it.

On many of the major version of the Unix operating system, compiling Zebra is a simple matter of typing `make` in the top-level distribution directory (this is the directory that was created when you executed `tar`). Normally, Zebra compiles cleanly at least on Linux, Digital Unix (DEC OSF/1), and IBM AIX. On certain platforms (such as SunOS), you will need to edit the top-level `Makefile` to set the `ELIBS` variable to include the Berkeley Socket Libraries. On other Unix platforms, you *may* need to modify Makefiles or header files, but in general, we have found Zebra to be easily portable across modern Unix-versions. You do need an ANSI-C compliant compiler (you'll see a long list of Syntax-errors during the compile if your default compiler is not ANSI C), but again, this is standard on most modern Unix-systems. If you don't have one, the freely available GNU C compiler is available for many systems.

3 The First GILS Database

Having successfully acquired the software, it's time to try it out. The directory `test` under the main distribution directory contains a small sample database of GILS records.

NOTE: The records included in the distribution are part of a sample set provided by the US Geological Survey, as a service to GILS implementors. They are included for testing and demonstrating the software, and neither the USGS or Index Data nor anyone else should be held responsible for their contents.

If you `cd` to the `test` directory, the first thing to notice is the file `zebra.cfg`. There has to be a file like this present whenever you run Zebra - it establishes various settings and defaults, and we'll return to its contents below (a detailed description is found in the general Zebra documentation file).

The subdirectory `records` contains the sample records. We'll get back to them, too.

The first order of business is to index the sample records, and create the access files required by the Z39.50 server. To do this, position yourself in the `test` directory, and type the command

```
$ ../index/zebraidx update records
```

The indexing program will respond with a stream of control information, and when it completes, the database is ready. To start the Z39.50 server, type the command `../index/zebrasrv`.

Assuming that nothing unfortunate happened, you are now running a GILS-compliant Z39.50 server on the port 9999 on your local machine (to learn how to run the server at a different port, and redirect the diagnostic output to a file, consult the section on *Running zebrasrv* in the general documentation). The database containing the sample records is named `Default`.

To test the server, you can use any compatible Z39.50 client. You can also use the simple demonstration client which is included with Zebra itself. To do this, start a new session on your machine (or put the server in the background). Change to the directory `yaz/client` under the main Zebra distribution directory. Now execute the command

```
$ ./client tcp:localhost:9999
```

If all went well, the client will tell you that it has established an association with your test server. To test it, try out these commands:

```
Z> find surficial
Z> show 1
```

The default retrieval syntax for the client is USMARC. To try other formats for the same record, try:

```
Z>format sutrs
Z>show 1
Z>format grs-1
Z>show 1
Z>elements B
Z>show 1
```

You can learn more about the sample client by reading the `README` file in the `yaz` directory.

4 The Records

The GILS profile is only concerned with the communication that takes place between two compliant systems. It doesn't mandate how the client application should behave, and it doesn't tell you how you should maintain and process data at the server side. Specifically, while the profile specifies a number of different exchange format for retrieval records.

For the purposes of this discussion, we will be using a simple, SGML-like representation of the GILS record structure. There is nothing magical or sacrosanct about this format, but it is easy to read and write, and because of its semblance of SGML and HTML, it is familiar to many people. If you would like to use a different, local representation for your GILS records, you can read the general Zebra documentation to learn how to establish a custom input filter for your particular record format.

In the SGML-like syntax, each record should begin with the tag `<gils>`. This selects the GILS profile, and provides context for the content tags which follow. Similarly, each record should finish with the end-tag `</gils>`.

The body of the record is made up by a sequence of tagged elements, reflecting the *abstract record syntax* of the GILS profile. Some of these elements contain simple data, or text, while others contain more tagged elements - these are complex, or constructed, data elements. The tag names generally correspond to the tag names provided in the GILS profile. Capitalization is ignored in tag names, as are dashes (-). Hence, `local-subject-index` is equivalent to `LocalSubjectIndex` which is the same as `LOCALSUBJECTINDEX`.

It is useful to look at the records in the `test/records` as examples of how SGML-formatted GILS record can look. Note that whitespace is generally ignored, so you can choose whatever layout of your records that suits you best.

5 The Zebra Configuration File

As mentioned, the Zebra indexer and server always look for the file `zebra.cfg` in their current working directory (unless they are told to look for it elsewhere with the `-c` option). The example file in the `test` directory represents all but the bare minimum for such a file. We find the following to be a powerful setup for a GILS-like database (everything preceded by (#) is ignored by the software):

```
#
# Sample configuration file for GILS database
#

# Where are the configuration files located?
profilePath: /usr/local/lib/zebra

# Load attribute sets for searching
attset bib1.att
attset gils.att

# Records are identified by their path in the file system
recordId: file

# Store information about records to allow deletion and updating
storeKeys: 1

# Records are structured
recordType: grs

# Where to store the indexes
register: /datadisk/index:500M

# Where to store temporary data while merging with register
shadow: /datadisk/shadow:500M
```

If you like, you can paste this file straight into a `zebra.cfg` file ready for your own use (with a bit of editing of the pathnames). In the following, we'll explain the individual settings. For the full story on the `zebra.cfg` file and the configuration options of Zebra, you should read the general documentation.

profilePath

This field tells Zebra where to look for the configuration files. In the distribution, these files are located in the `tab` directory, but you may wish to put them someplace else for convenience. If necessary, you can provide multiple directory paths, separated by (:).

attset

This field tells the Zebra server which attribute sets it should support for searching. You could get by with just loading the GILS set, but if you load BIB-1 as well, Zebra will support both sets for those GILS attributes that are inherited from BIB-1.

recordId

The `recordId: file` setting tells Zebra that individual records should be identified by the physical files in which they are located. In this mode, your database will always (after an update operation) reflect the contents of the directory (or directories).

storeKeys

This setting tells Zebra to store additional information about each record, to facilitate updating. In combination with the `recordId: file` setting, this is a very convenient maintenance option. If you maintain your records as individual files in a directory tree, you have only to run `zebraidx` with the top-level directory as an argument. If new files are added, they are entered into the database. If they are modified, the indexes are changed accordingly, and if they are deleted from the filesystem (or renamed), the indexes are also updated correctly, the next time you run `zebraidx`.

recordType

This setting selects the type of processing which is to take place when a record is accessed by the indexer or the Z39.50 server. GRS stands for *Generic Record Syntax*, and signals that the records are structured.

register

In the first test above, you may have noticed that the `zebraidx` created a number of files in the working directory. Some of these files, which contain the indexing information for the database, can grow quite large, and it is sometimes useful to place them in a separate directory or file system. You should provide the path of the directory followed by a colon (:), followed by the maximum amounts of megabytes (M) or kilobytes (K) of disk space that Zebra is allowed to use in the given directory. If you specify more than one directory:size combination *on the same line*, Zebra will fill up each directory from left to right. This feature is essential if your database is so large that the registers cannot fit into a single partition of your disk.

shadow

The format of this setting is the same as for the one above. If you provide one or more directory for the "shadow system", you enable the safe updating system of the Zebra indexer. When changes to the records are merged into the register files, the files are not changed immediately. Instead, the changes are written into separate files, or "shadow files". At the end of the merging process, or in a separate operation, the changes are "committed", and written into the register files themselves. This final step is carried out by the command `zebraidx commit` - the `commit` directive can also be given on the same command line as the `update` directive - at the end of the command line. The shadow file system can consume a lot of disk space - particularly in a large update operation which involves almost all of the index, but the benefits are substantial. If the system crashes during an update procedure, or the process is otherwise interrupted, the registers are left in an unknown state, and are effectively rendered useless - this can be unfortunate if the index is very large, but the use of the shadow system greatly reduces the risk of an index being damaged in this way. Further, when the shadow system is enabled, your clients may access the Zebra server without interruption throughout the update and commit procedures - Zebra will ensure that the parts of the register accessed by the server are always consistent.

6 Creating Your Own Database

Whenever we create a new database with Zebra, we find it useful to first set up a new, empty directory. This directory will contain the configuration file, the lock files maintained by Zebra (unless you specify a different location for these), and any logs of updates and server runs that you may wish to keep around.

The first thing to do is set up the `zebra.cfg` file for your database. You can copy the one from the `test` directory, or you can create a new one using the example settings described in the previous section. Once you get your server up and running, you may want to read the description of the `zebra.cfg` file in the general documentation, to set up additional defaults for database names, etc.

If you copy one of these files, you should be careful to update the pathnames to reflect the setup of your own database. In particular, if you want to specify one or more directories for the register files and/or the shadow files, you should make sure that these directories exist and are accessible to the user ID which will run the Zebra processes.

You need to make sure that your GILS records are available, too. For small to medium-sized (say, less than 100,000 records) databases, it is sometimes preferable to maintain the records as individual files somewhere in the file system. Zebra will, by default, access these files directly whenever the user requests to see a specific record. However, you can set up Zebra to maintain the database records in other ways, too. Consult the general documentation for details.

Finally, you need to run `zebraidx` to create the index files, and start up the server, `zebrasrv` (the server can be run from the `inetd` if required), and you are in business.

To access the data, you can use a dedicated Z39.50 client, or you can set up a WWW/Z39.50 gateway to allow common WWW browsers to search your data. CNIDR's Isite package includes a good, free gateway that you can experiment with.